## 10 Objekter

De programmer, der har været vist i lektion 1 – 9 har ikke adskilt sig særligt fra gammeldags sekventiel programmering. Vi har primært brugt simple datatyper som heltal og decimaltal.  
Java-programmeringssproget kaldes **objektorienteret** sprog, fordi det benytter sig af **objekter**.  
I denne lektion tager vi fat på begrebet og ser på kendte og nye objekter.

Objekter modellerer den verden vi befinder os i. Vi omgiver os med ting og begreber.  
F.eks. kan en **bold** være et **objekt**. Den kan have en farve, en diameter, en elasticitet. Dette kan vi kalde **boldens egenskaber**.  
En bold kan også udføre handlinger: Den kan hoppe, trille, støde, svæve - med og uden skruning.   
Dette kan vi kalde **boldens metoder**.

Fordelen ved at bygge programmer ved hjælp af objekter er, at vi til enhver tid kan tilpasse objekterne til vores foranderlige verden. Vi kan tilføje nye **egenskaber** til bolden: RadiosenderNr.  
og nye **metoder**: udsendSignal( ); Disse ændringer kan vi lave uden at fjerne eller ændre de gamle.  
Således kan gamle programmer, der bruger bolde fortsat fungere, - og nye programmer er ikke begrænset af fortidens få muligheder.

**Objekter** er ikke begrænset til **håndgribelige ting** som bolde. **Objekter** kan også modellerer f.eks. **begreber** som en ”fejlmelding” eller ”humør”.

### 10.1 Objekter: Ikke-simple variable

En variabel er enten en **simpel værdi** eller en **reference** til et objekt.  
Ligesom de simple datatyper skal erklæres før man kan tage dem i brug, så skal objekter også erklæres:

{ int tal;

String mit\_objekt; }

Der er nu reserveret plads til et heltal *int tal*; og der er reserveret plads til et objekt af typen *String*. Navnet *mit\_objekt* fungerer som en reference til objektet: Det er som adresse-sedlen på en kuffert, - den angiver hvor kufferten bor. Dvs. referencen angiver hvor i hukommelsen objektet kan findes.   
Indtil nu har vi kun erklæret, at vi vil have et heltal med varia­bel­-nav­net *tal* og et objekt af typen *String* med objektnavnet *mit\_objekt*. String-typen er ikke en simpel datatype men derimod en klasse, der definerer objekter.   
Objektet *mit\_objekt* er ikke skabt endnu, dette sker først når vi bruger new-operatoren:

{ *mit\_objekt = new* String(”Ofelia, oh min elskede”); }

Først nu er der skabt et eksemplar af klassen/typen *String*. Referencen til dette eksemplar er variabel-navnet *mit\_objekt.* Dette refererer til en plads i hukommelsen hvor teksten ”Ofelia, oh min elskede” er gemt. String er ikke et Java-reserveret ord, da det er navnet på en klasse der er skrevet af program­mører.   
… og så det med småt:   
 Vi kan også skabe en tekst-streng ved blot at skrive *String* *mit\_objekt = ” Ofelia, oh min elskede” ;*  
altså direkte uden *new*-operatoren. Det er fordi String er så grundlæggende en type at der er gjort en undtagelse, der letter programmeringen…

I de tidligere lektioner har vi stiftet bekendtskab med klassen *System* som vi brugte til at lave objektet *out* med for at kunne skrive til skærmen.

Vi benytter klasser andre har skrevet, så vi slipper for selv at beskæftige os med detal­jer­ne. *System* tager sig af at kontakte operativsystemet på computeren og få dette til at skri­ve tekststrengene på skærmen. Vi behøver ikke at kende noget som helst til opera­tiv­sy­ste­mer for at kunne bruge objektet System.out’s metoder.

*{ System.out.print(”Nu er vi i gang og skriver til skærmen”);* }

*System.out* repræsenterer et **objekt** *out* fra **klassen** *System* som er skrevet af en programmør og stillet til vores disposition. *System.out*-objektet (static PrintStream out, se: <http://docs.oracle.com/javase/6/docs/api/java/lang/System.html> ) stiller forskellige **metoder** til rådighed fx *print( )* og *println( )*. Metoderne skriver default til skærmen.

<<class>>

CylinderBeregn

<<class>>

System

<<object>>

System.out

println

print

main

Dit eget program Skrevet af andre, men   
offentligt tilgængeligt

### 10.2 Klasser og objekter

Af få skabt et objekt svarer til, at vi beder om at få bygget et typehus på en bestemt grund.

En arkitekt eller entreprenør har tegnet huset. Tegningerne vil svare til en **klasse** i Java.  
Selvom vi har tegningerne, har vi intet hus endnu.

Først når huset er bygget, findes der et **eksemplar** af typehuset. I Java bruger vi *new-*operatoren for at få et eksemplar – et **objekt** af en klasse.

Der kan godt skabes mange typehuse, men de må ligge på hver sin byggegrund. Således kan der skabes mange objekter, f.eks. af typen String, tekststrenge, men de må ligge på hver sin hukommelsesplads i computeren. Se tegningen herunder.

<<class>>

String

<<object>>

mit\_objekt

<<object>>

navn

<<object>>

tekst

Type/klasse

Type/tegning

Eksemplarer/ objekter

|  |
| --- |
| Hukommelsesstak |
| **”Ofelia, oh min …”** |
| ”Hans Hansen” |
| **” Skriv på skærm”** |
|  |
|  |
|  |
|  |

Eksemplarer/Huse

**Svalevej 15**

Adresser

Adresser

**Harevej 2**

Vi behøver ikke at kende noget til arkitektur eller vand- og varmeforsyning for at få bygget et typehus, blot skal vi kende en bygmester, der har tegningerne, for at få bygget et (forhåbentlig) velfungerende hus.

Sådan er det også i Java: Vi kan få lavet de **objekter** vi har brug for, blot vi kender **klassen** og har adgang til kildeteksten til denne.

### 10.3 Vores egen møntfabrik

Nu har vi set på andre programmørers objekter *String* og *System.out*. Lad os selv lave et objekt der skal modellerer en mønt. Bagefter laver vi et program, der slår plat og krone med mønten.

package mitfoersteprojekt;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Moent.java Forfattere: Lewis and Loftus tilrettet af Jette Levison

// Repræsenterer en mønt med to sider og som kan kastes.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class Moent

{

public final int KRONE = 0;

public final int PLAT = 1;

private int face;

//-----------------------------------------------------------------

public Moent ()

{ // Konstruktør der danner en mønt som straks kastes.

System.out.println(" test: Moent: Hej, jeg er en ny mønt. Nu kaster jeg mig lige, så jeg er klar...");

flip();

System.out.println(" test: Moent: "+this); //”this” forklares i lektion 11

}

//-----------------------------------------------------------------

public void flip ()

{ // Kaster mønten ved at vælge siden tilfældigt.

face = (int) (Math.random() \* 2);

}

//-----------------------------------------------------------------

public int getFace ()

{ // Returnerer den side der vender op i form af et heltal.

return face;

}

//-----------------------------------------------------------------

public String toString()

{ // Returnerer den gældende side i form af en tekststreng.

String faceName, txt;

if (face == KRONE)

faceName = "krone";

else

faceName = "plat";

txt = "Jeg er en mønt og viser lige nu "+faceName;

return txt;

}

//-----------------------------------------------------------------

}

”Programmet” herover kan ikke køres, da der ikke er en main-metode. Der er nu lavet en **klasse** der definerer en mønt. Mønten har en side: plat eller krone, der vælges tilfældigt.

For at afprøve vores mønt må vi lave en driver-klasse med en main-metode.

package mitfoersteprojekt;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// OpsumPlatKrone.java Oprindelige forfattere: Lewis and Loftus

// Modereret af: Jette Levison

// Demonstrerer brugen af programmør-defineret klasse.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public class OpsumPlatKrone

{

//-----------------------------------------------------------------

// Kaster en mønt adskillige gange og tæller antallet af krone- og

// platsider.

//-----------------------------------------------------------------

public static void main (String[] args)

{

final int NUM\_FLIPS = 1000;

int sum\_krone = 0, sum\_plat = 0;

System.out.println("\ntest: Main-program: Om lidt laves en mønt");

Moent minMoent = new Moent(); // instantierer Moent-objektet

System.out.println(minMoent.toString());//test-udskrift

System.out.println("test: Main-program: ok, nu skal vi kaste mønten "

+NUM\_FLIPS+" gange.");

for (int count=1; count <= NUM\_FLIPS; count++)

{

minMoent.flip();

if (minMoent.getFace() == minMoent.KRONE)

sum\_krone++;

else

sum\_plat++;

}

System.out.println ("\nAntal af kast var: "+NUM\_FLIPS+". Fordelt således");

System.out.println ("\tantallet af kroner var: " + sum\_krone);

System.out.println ("\tantallet af plat var: " + sum\_plat + "\n");

}

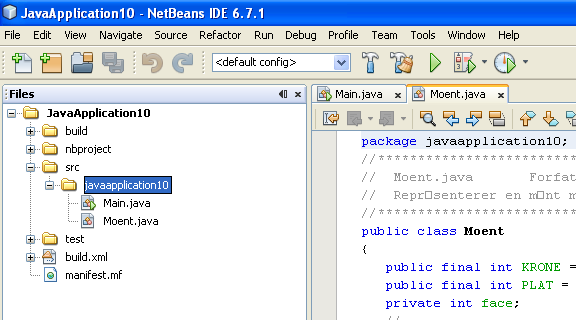
}

I dette program skaber vi en mønt *minMoent,* det kaldes at instantiere objektet:   
 *Moent minMoent = new Moent();*   
Herefter kan vi kalde metode i objektet ved at skrive *minMoent.flip();*Endelig kalder vi metoden *minMoent.getFace()* og sammenligner returværdien med konstanten *minMoent.KRONE*.

**Opgave 10:**

I arkivet på Fronter ligger en mappe med ”programmerne” til denne lektion.  
Der er i alt fire klasser, hvoraf to kan køres (main-metode) og to er rene objekt-klasser, der ikke kan køres selvstændigt.

1. Kopier .java filerne og højreklik i Netbean under filer til venstre på pakkenavnet under mappen src og sæt ind. Så klarer Netbean resten.
2. Kør programmet OpsumPlatKrone og bliv fortrolig med dets virkemåde.



Bemærk package navnet ændres automatisk til det korrekte. Ellers kan refactor nogle gange klare problemerne.

Dvs. der er ingen ting at aflevere til denne lektion. Men måske du mangler en øvelse eller to fra lektion 6-9?